



Rédiger ses **spécifications** avec Spock

“Ecrire des tests ? C’est illogique.”



Faisons connaissance



Adrien Bonnin

#ippontech #cleancode
#spock #grails #micronaut
#salsa #bachata
#retrogaming #mtga



@adbonnin



www.linkedin.com/in/adbonnin



<https://github.com/adbonnin>



abonnin@ippon.fr



Specification + Mock
=
Spock



Créé en 2008 par Peter Niederwieser
Développeur chez Gradleware

Second commiteur : Luke Daley, créateur de GEB
Développeur chez Gradleware



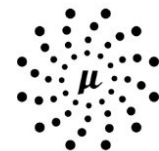
Utilisé par :



apache
tapestry
Code less, deliver more.



mongoDB®



MICRONAUT



NETFLIX

Ippon Technologies 2019

Qui sait **développer** en **Groovy** ?



Groovy c'est bon, mangez en !

Apprentissage rapide

99% du code Java 1.7 est compatible

Typage fort et dynamique

Déclarations simplifiées

Groovy Truth

Expressions vraies ou fausses

Surcharge des opérateurs

Redéfinition du comportement de
+, -, *

Interopérabilité avec la JVM

Compatible Java, Kotlin, JavaScript, etc...

AST

Abstract Syntax Tree

Manipulation du code



```
apply plugin: 'groovy'

repositories {
    mavenCentral()
    jcenter()
}

dependencies {
    compile 'org.codehaus.groovy:groovy-all:2.4.10'

    // Test dependencies
    testCompile
    'org.spockframework:spock-core:1.0-groovy-2.4'
    testCompile 'cglib:cglib-nodep:3.2.6'
    testCompile 'org.objenesis:objenesis:2.6'
}
```



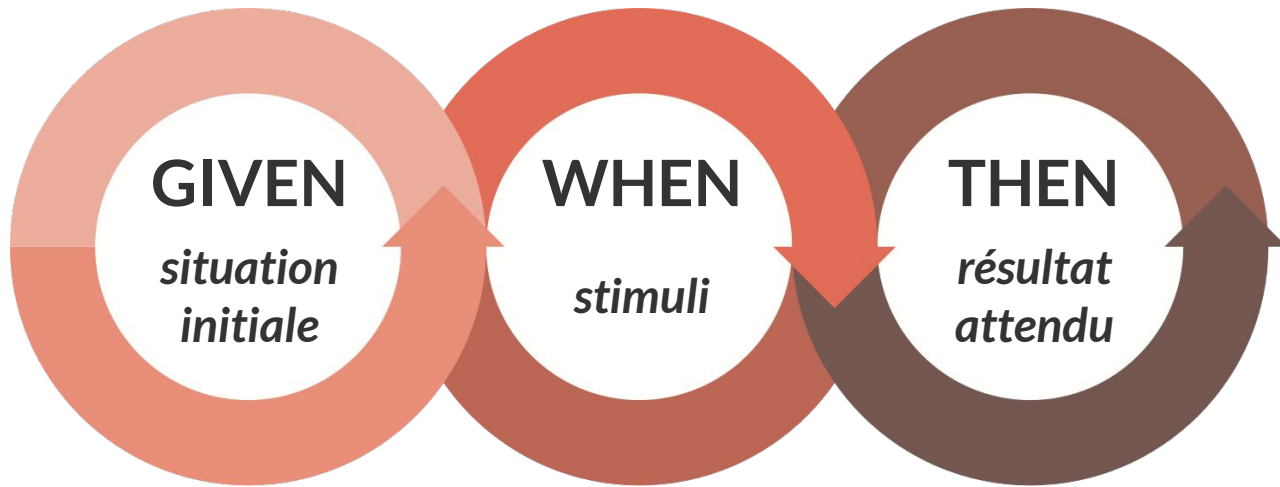
```
<plugin>
  <groupId>org.codehaus.gmavenplus</groupId>
  <artifactId>gmavenplus-plugin</artifactId>
  <version>${gmaven-plugin.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>testCompile</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

[...]

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>${surefire-plugin.version}</version>
  <configuration>
    <useFile>false</useFile>
    <includes>
      <include>**/*Spec.java</include>
    </includes>
  </configuration>
</plugin>
```



Dan North: Concepteur du Behaviour-Driven Development




```
class MathSpec extends Specification {  
    void "doit retourner le max de 2 nb"() {  
        given:  
        def a = 1  
        def b = 2  
  
        when:  
        def max = Math.max(a, b)  
  
        then:  
        max == 2  
    }  
}
```

*Ceci n'est pas un test...
mais une spécification !*

- specification
- feature

- pas de bruit
- méthode en toutes lettres
- étend une classe abstraite
- séparation en blocs
- pas d'asserts ! exp bool

**sonarqube**

JUnit runner Sputnik

=

Interopérable avec les IDE,
outils de build et
serveurs d'intégration

```
class MathSpec extends Specification {
```

```
  void "doit retourner le max de deux nombres"() {
```

```
    when:
```

```
    def result = Math.max(a, b)
```

```
    then:
```

```
    result == max
```

```
    where:
```

```
    a | b | | max
```

```
    1 | 2 | | 2
```

```
    2 | 4 | | 4
```

} Les variables sont
déclarées après leurs
utilisation

```
  }  
}
```

Tests OK

✓ MathSpec

✓ MathSpec.doit retourner le max de deux nombres

Tests KO

✗ MathSpec

✗ MathSpec.doit retourner le max de deux nombres



```
class MathSpec extends Specification {  
  
  @Unroll("doit retourner #max pour le max entre #a et #b")  
  void "doit retourner le max de deux nombres"() {  
    when:  
    def result = Math.max(a, b)  
  
    then:  
    result == max  
  
    where:  
    a | b | | max  
    1 | 2 | | 2  
    2 | 4 | | 2  
  
  }  
}
```

Condition not satisfied:

Math.max(a, b)	==	max
4 2 4		2
		false

Sans @Unroll

✗ MathSpec
✗ doit retourner le max de deux nombres

Avec @Unroll

✗ MathSpec
✓ doit retourner 2 pour le max entre 1 et 2
✗ doit retourner 2 pour le max entre 2 et 4

Monsieur Spock, raconte moi une spécification

1

```
@Title("Spécification relative aux listes")
@Narrative("Une description encore plus longue")
class ListSpec extends Specification {
```

2

```
@Issue("CD-42")
void "doit ajouter une valeur à une liste"() {
```

3

```
given:
@Subject def list = []
```

```
expect: "la liste est vide"
list.isEmpty()
```

```
when:
list.add(value)
```

```
then: "la liste n'est plus vide"
!list.isEmpty()
list.size() == 1
```

5

```
and: "la valeur a été ajoutée"
list.first() == value
```

```
where:
value = 1
```

```
}
```

```
}
```

```
when:
def isEmpty = list.isEmpty()

then:
isEmpty
```

```
expect:
list.isEmpty()
```

Spécification relative aux listes

Une description encore plus longue

Features:

- doit ajouter une valeur a une liste

doit ajouter une valeur a une liste

Issues:

- CD-42

Given: ----
Expect: la liste est vide
When: ----
Then: la liste n'est plus vide
And: la valeur a ete ajoutee
Where: ----

Examples:

value

1

OK

1. titre & description
2. référence du bug
3. sujet de la feature
4. commentaire du bloc
5. liaison entre les blocs

```
class ListSpec extends Specification {  
    @Subject def list = []  
  
    void "doit ajouter une valeur à une liste"() {  
        expect:  
        list.isEmpty()  
  
        when:  
        list.add(value)  
  
        then:  
        !list.isEmpty()  
        list.size() == 1  
  
        and:  
        list.first() == value  
  
        where:  
        value = 1  
    }  
}
```

Étanchéité et Tests idempotents

```
class IdempotentSpec extends Specification {  
  
    @Shared  
    def ds = new JdbcDataSource(url: 'jdbc:h2:mem:test')  
  
    @AutoCleanup  
    def connection = ds.getConnection()  
  
    [...]  
}
```

```
class LifeCycleSpec1 extends Specification {
```

```
  @Shared @AutoCleanup
```

```
  def sharedVariable = new Closeable() {  
    { println('> shared variable') }  
    void close() { println('< cleanup shared variable') }  
  }
```

Partagée entre les features

```
  @AutoCleanup
```

```
  def variable = new Closeable() {  
    { println(' > build variable') }  
    void close() { println(' < cleanup variable') }  
  }
```

Instanciée pour chaque feature

```
  def "doit présenter le cycle de vie"() {  
    given:  
    println(" > setup feature $index")
```

Appelé à chaque itération

```
    expect:  
    true
```

```
    cleanup:  
    println(" < cleanup feature $index")
```

```
    where:  
    index << [0, 1]
```

```
  }  
}
```

Résultat

```
> shared variable  
> build variable  
  > setup feature 0  
  < cleanup feature 0  
< cleanup variable  
> build variable  
  > setup feature 1  
  < cleanup feature 1  
< cleanup variable  
< cleanup shared variable
```



```
class LifecycleSpec2 extends Specification {  
  
  void setupSpec() { println('> setupSpec') }  
  
  void cleanupSpec() { println('< cleanupSpec') }  
  
  void setup() { println(' > setup') }  
  
  void cleanup() { println(' < cleanup') }  
  
  def "doit présenter le cycle de vie"() {  
    given:  
      println(" > setup feature $index")  
  
    expect:  
      true  
  
    cleanup:  
      println(" < cleanup feature $index")  
  
    where:  
      index << [0, 1]  
  }  
}
```

Appelé une fois

Appelé à chaque feature

Appelé à chaque itération

Résultat

```
> setupSpec  
> setup  
> setup feature 0  
< cleanup feature 0  
< cleanup  
> setup  
> setup feature 1  
< cleanup feature 1  
< cleanup  
< cleanupSpec
```

```
interface PersonnelDao {  
    int compterPersonnels()  
    int persisterPersonnel(Personnel personnel)  
}
```



Initialiser
la base de données



Mettre à jour
les personnels



Vérifier
le nombre de
personnels



Libérer
les ressources



Itérer
sur les cas de tests

```
class PersonnelDaoSpec extends Specification {
```

```
  @AutoCleanup def sql = Sql.newInstance('jdbc:h2:mem:test')
```



Libérer
les ressources

```
  @Subject def personnelDao = new PersonnelDaoImpl(sql)
```

```
  void setup() {
```

```
    sql.execute("""create table personnel (id identity not null primary key,  
                                           nom varchar(25) not null)""")
```

```
  }
```

```
  @Unroll
```

```
  void "doit persister les personnels #nomPersonnels"() {
```

```
    when:
```

```
    nomPersonnels.each { nom ->
```

```
      personnelDao.persisterPersonnel(new Personnel(nom: nom))
```

```
    }
```

```
    then:
```

```
    personnelDao.compterPersonnels() == expectedNombrePersonnels
```

```
    where:
```

```
    nomPersonnels      || expectedNombrePersonnels
```

```
    ['Spock']           || 1
```

```
    ['Spock', 'Kirk']   || 2
```

```
  }
```

```
}
```



Initialiser
la base de données



Mettre à jour
les personnels



Vérifier
le nombre de personnels



Itérer
sur les cas de tests

Quand doit-on utiliser
un **Stub**, un **Mock** et un **Spy** ?



```
given:  
def builder = Stub(Builder) {  
  build() >> {  
    return "abc"  
  }  
}
```

```
given:  
def builder = Mock(Builder)  
  
[...]  
  
then:  
3 * builder.build() >> "abc"
```

```
given:  
def builder = Spy(BuilderImpl) {  
  2 * build() >> {  
    return callRealMethod()  
  }  
}
```

Les interactions sont déléguées à un objet différent.



Stub

- Retourne un résultat préétabli



Mock

- Retourne un résultat préétabli
- Vérifie les appels de méthodes



Spy

- Intermédiaire
- Peut retourner le vrai résultat
- Vérifie les appels de méthodes

```
interface TeleporteurService {  
  
    /**  
     * Téléporte un personnel par son nom.  
     *  
     * @param nom le nom du personnel à téléporter  
     * @return true si au moins un personnel a été téléporté, sinon faux  
     */  
    boolean transporter(String nom)  
}
```

```
interface PersonnelDao {  
  
    List<Personnel> rechercherPersonnels(Recherche recherche)  
  
    static class Recherche {  
  
        String nom  
    }  
}
```

```
interface Teleporteur {  
  
    boolean teleporter(List<Personnel> personnels)  
}
```




```
class TransporteurServiceSpec extends Specification {
```

1

```
  def mockTransporteur = Mock(Transporteur)
  def mockPersonnelDao = Mock(PersonnelDao)
```

```
  @Subject
```

```
  def service = new TransporteurService(mockTransporteur, mockPersonnelDao)
```

```
  def "doit transporter les personnels"() {
```

```
    when:
```

```
    def result = service.transporter('Spock')
```

3

```
    then:
```

```
    1 * mockPersonnelDao.rechercherPersonnels(_) >> personnels
```

2

4

```
    then:
```

```
    expectedTransporter * mockTransporteur.transporter(personnels) >> transportReussi
```

```
    and:
```


```
    result == expectedResult
```

```
  where:
```

personnels	transportReussi	expectedTransporter	expectedResult
[]	<code>false</code>	<code>0</code>	<code>false</code>
[new Personnel(nom: 'Spock')]	<code>false</code>	<code>1</code>	<code>false</code>
[new Personnel(nom: 'Spock')]	<code>true</code>	<code>1</code>	<code>true</code>

```
}
```

1. définition des mocks
2. wildcard dans la conf
3. nombre d'appels
4. ordre des appels

Cas d'utilisations	Outil(s) à domicile	Challenger
Behavior Driven Design	Cucumber JBehave	Spock, <i>what else ?</i> Tests asynchrones Intégration Spring 
Tests unitaires	JUnit TestNG	
Mock, Stub et Spy	EasyMock jMock Mockito PowerMock jMockit	



CONCLUSION

Rédiger ses **spécifications** avec
Spock c'est...

**Facile à lire,
Rapide à comprendre,
Simple à maintenir.**



Discovery to Delivery

Ippon.fr

contact@ippon.fr

+33 1 46 12 48 48



@IpponTech